

# Parallelization of the Naval Space Surveillance Satellite Motion Model

Warren E. Phipps Jr.

Beny Neta\*

D. A. Danielson

Naval Postgraduate School  
Department of Mathematics  
Code MA/Nd  
Monterey, CA 93943

---

\*Author to whom correspondence should be addressed.

## Abstract

The Naval Space Surveillance Center (NAVSPASUR) uses an analytic satellite motion model based on the Brouwer - Lyddane theory to track objects orbiting the Earth. In this paper we develop several parallel algorithms based on this model. These have been implemented on the INTEL iPSC/2 hypercube multi-computer. The speed-up and efficiency of these algorithms will be obtained. We show that the best of these algorithms achieves 87% efficiency if one uses a 16-node hypercube.

## Introduction

The Naval Space Surveillance Center (NAVSPASUR) uses an analytic satellite motion model to track objects orbiting the Earth. This model is implemented in the Fortran subroutine PPT2. This subroutine predicts an artificial satellites's position and velocity vectors at a selected time to aid in the tracking endeavor. Several calls to the subroutine may be required to aid in the identification of one object. A substantial increase in the number of objects or a desire to increase the accuracy of the model will require a similar increase in computer time. Parallel computing offers one option to decrease the computation time without sacrificing accuracy.

For a multicomputer, the user must partition the problem among the processors. Two decompositions are possible and will be discussed here, control decomposition and domain decomposition.

In this paper we determine the parallel computing potential of the current NAVSPASUR model as applied to a MIMD computer and simulated on an iPSC/2 hypercube. In the next section, we develop a control decomposition method and discuss the speed-up attained by our numerical experiments on a 4-node hypercube. In section 3, we discuss a domain decomposition method. We show that domain decomposition yields higher speed-up. We also develop a model showing that 16 nodes yield optimal efficiency (almost 90%) and discuss how to utilize larger dimension hypercubes without losing efficiency.

## Control Decomposition

Control decomposition is the strategy of dividing tasks among the nodes. This is recommended for problems with irregular data structures or unpredictable control flows (see Parallel Programming Primer pp. 4-6 in [5]). The exact tasks required of each node are explicitly stated in the parallel program.

In order to predict a satellite's state vector considering the secular and periodic correction terms due to the zonal harmonics and a correction term for each element due to the sectoral harmonics, the NAVSPASUR model requires the completion of 55 major tasks. These tasks are described by Phipps (1992). The first step in partitioning these tasks among the nodes

Table: 1 Concurrent NAVSPASUR Orbit Model Tasks

Level															
1	T2 (2.47-8)	$a_0''$ (2.50)													
2	$\gamma_1'$ (2.54)	$\dot{a}$ (2.41)	$\dot{e}$ (2.45)	$\Delta\ell$ (2.38)											
3	$dg/d\ell$ (2.51)	$dh/d\ell$ (2.52)	$\ell''$ (2.55)	$a''$ (2.57)	$e''$ (2.57)	VLE1 (2.59)	VLE2 (2.59)	VLE3 (2.59)	VLH1I (2.59)	VLH2I (2.59)	VLH3I (2.59)	VLS1 (2.60)	VLS2 (2.60)	VLS3 (2.60)	
4	$g''$ (2.55)	$h''$ (2.55)	$\cos f''$ (2.58)	VLL2 (2.59)											
5	$\delta_1 e$ (2.59)	$e \delta_1 \ell$ (2.59)	$r''$ (2.58)	$\sin f''$ (2.58)	$\sin I'' \delta_1 h$ (2.59)	$\delta_1 z$ (2.67)	$\delta_2 e$ (2.61)	$e \delta_2 \ell$ (2.63)	$\sin I'' \delta_2 h$ (2.65)	$\delta_2 a$ (2.18)	sectoral (LUNAR)				
6	$\delta_1 I$ (2.59)	$e \delta \ell$ (2.64)	$\sin(I/2) \delta h$ (2.66)	$\delta_2 z$ (2.67)	$\delta e$ (2.62)	$a$ (2.69)									
7	$\delta I$ (2.64)	$e$ (2.70)	$\ell$ (2.70)	$z$ (2.68)											
8	$\cos I$ (2.70)	$h$ (2.70)	$\cos f$ (2.72)												
9	$g$ (2.71)	$r$ (2.72)													
10	$\hat{r}$ (2.37)	$\hat{v}$ (2.37)													
11	$\hat{r}'$ (2.37)	$\hat{v}'$ (2.37)													

was to determine which tasks could be completed concurrently. Concurrency was determined by the development of a hierarchy of the formulas used by the NAVSPASUR model. Each of the individual tasks were listed with its respective required input. Tasks which could be executed concurrently were listed on the same row of Table 1.

Remark: The equation numbers in the table refer to Phipps (1992) .

From this table, one can see that the number of tasks that could be computed concurrently at each level ranges from 2 to 14. Additionally, the computational requirements vary considerably among the tasks. for example, the computational requirement for the solution of Kepler's equation by Steffensen's method depends on the number of iterations necessary to achieve convergence. This variance in the number of operations required by the various tasks presented a potential problem in load balancing. In other words, bottlenecks are due to the fact that nodes are awaiting to receive results from computations performed by other processors. It was shown by Phipps (1992) that a manager-worker algorithm (to achieve load balancing) will increase the communication and thus decrease efficiency. Thus pre-scheduling of tasks is done. The optimal number of nodes is found to be four. In Table 2, we list the tasks scheduled for each node. A computer program,  $P^3T - 4$ , was developed for the hypercube. Experiments with this program show that the computation time ( $t_c$ ) for  $P^3T - 4$  is about half that of PPT2. Unfortunately, the communication time ( $t_m$ ) was so high that the total time for  $P^3T - 4$  was larger (see table 3).

One method to reduce the ratio of communication to computation is by computing the path of  $n$  satellites at the same time. In other words, currently the program PPT2 reads the initial values of one satellite and computes its position at a given time, and then moves on to the computation of the next satellite position. Since each communication requires an overhead, it is cheaper to send a long message. To arrange that, we suggest that the program reads initial values of several satellites and computes the paths concurrently. This will require the same number of messages, but each one is  $n$  times longer. The efficiency,

Table: 2 Tasks for each node

Node 0	Node 1	Node 2	Node 3
Recover $a''$ 137 flops send 8 bytes		Compute $T_2$ 113 flops send 8 bytes	
Compute secular corrections - $\ell, a$ , and $e$ 45 flops send 24 bytes	Compute secular correction - $g$ 80 flops send 8 bytes	Compute long period correction - $z$ 113 flops	Compute secular correction - $h$ 85 flops send 8 bytes
Compute long period correction - $\ell$ 63 flops	Compute long period corrections - $e$ and $I$ 64 flops	Solve Kepler's Equation $\sim 308$ flops	Compute sectoral terms 528 flops send 48 bytes
Compute short period correction - $\ell$ 46 flops send 8 bytes	Compute short period corrections - $e$ and $I$ 88 flops send 16 bytes	Compute short period correction - $z$ 14 flops send 8 bytes	Compute long period correction - $h$ 69 flops
Compute short period correction - $a$ 24 flops			Compute short period correction - $h$ 52 flops send 24 bytes
Solve Kepler's Equation $\sim 308$ flops			
Collect all terms			
Compute state vector 74 flops			

Table: 3 **P<sup>3</sup>T – 4** Execution Time Breakdown

Algorithm	$t_c$ (milliseconds)	$t_m$ (milliseconds)	$t_1$ (milliseconds)
<b>PPT2</b>			
(one node)	11.2	NA	11.2
<b>P<sup>3</sup>T – 4</b>			
node 0	4.3	19.0	23.3
node 1	2.2	15.9	18.1
node 2	2.7	14.7	17.4
node 3	5.8	15.7	21.5

$E_p^n$ , is given by

$$E_p^n = \frac{nt_1}{p(nt_c + t_m)}$$

since the communication time is not affected by the length of the message. As one increases the number  $n$ , the limit is

$$\lim_{n \rightarrow \infty} E_p^n = \frac{t_1}{pt_c}.$$

Using the values in the above table one finds that the efficiency is bounded by 0.49. This is the best we were able to achieve. The reason is that the computation time for 1 satellite is 5.8 sec on 4 nodes and 11.2 sce on 1 node. Thus the maximum achievable efficiency is bounded by .5. Since this is not high enough, we have tried domain decomposition. This is discussed in the next section.

## Domain Decomposition

The strategy of domain decomposition is to reduce the computation time by the concurrent computation of several satellites' state vectors. Each node of the hypercube would complete identical tasks on different satellite data sets, simultaneously.

Unlike the application of the control decomposition strategy, the application of the domain decomposition strategy to the NAVSPASUR model was seemingly less arduous. First, because each node propagates satellite data sets independent of the other nodes, there exists no requirement for communication or synchronization among the nodes. This lack of communication simplifies the load balancing and sequential bottleneck problems present in the  $P^3T - 4$  parallel algorithm.

Second, because each node may perform the satellite state vector prediction tasks serially, the existing subroutine *PPT2* may be used with only minor modifications. Developing a parallel algorithm for predicting an individual satellite's state vector was a major task for the control decomposition strategy. Additionally, by using the existing *PPT2* code, the other

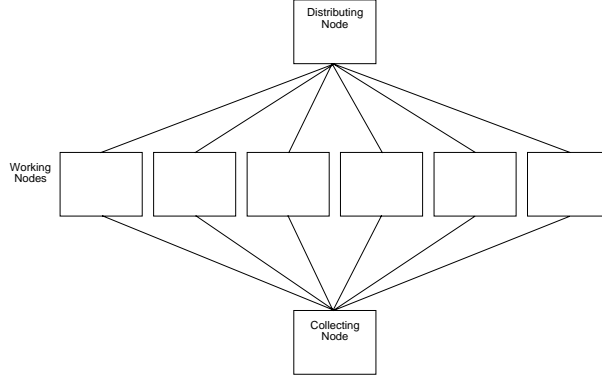


Figure 1:  $P^3T$  Algorithm

tasks completed by  $PPT2$  may be requested by the user using the same control variables as used by the original  $PPT2$  subroutine. The  $P^3T - 4$  program set was restricted to only predicting a satellite's state vector.

Finally, by using the serial subroutine  $PPT2$ , this strategy may be reduced to only developing an algorithm to distribute the data in a timely manner. Maximum efficiency will be achieved if the nodes do not have to wait for satellite data to propagate.

Intuitively, this strategy seems perfectly parallelizable. Although the various tasks performed by  $PPT2$  require different computation times, the total execution time for each node will be essentially the same if it is assumed that the various tasks are randomly distributed throughout the input data sets. The concern for this algorithm was the potential sequential bottlenecks at input/output portions of the program set. Reading and writing to external files can be very time consuming. In addition to the actual time spent reading/writing to an external file, a certain amount of time is spent to access the file. In order to minimize this time, the number of calls to read/write to a file should be minimized.

With the specific  $iPSC/2$  hypercube available, input/output is completed sequentially. Each node must compete with the other nodes to read and write to external files. To minimize time lost to accessing the file cataloging the set of satellites, a node was devoted to both the reading/distributing of input satellite data and to the collecting/writing of the results. The idea of using a single node to read the data and a single node to subsequently write the output is simple to implement and proved to be fastest method to overcome the bottlenecks with the input/output. The remaining nodes of the hypercube implement the NAVSPASUR model using a slightly modified  $PPT2$ . The diagram in Figure 1 depicts how the satellite data is distributed. The cost of using this simple algorithm to distribute and collect the data is the loss of two nodes. The only restriction on the size of the hypercube required by  $P^3T$  is that the attached cube must contain at least four nodes to achieve any speedup.

The graph in Figure 2 depicts the mean execution time for  $P^3T$  versus the number of satellites propagated using hypercubes of four and eight nodes respectively.  $P^3T$  was

Table: 4 Speedup and Efficiency Comparison

$P^3T$	# of satellites	$S_p$	$E_p$
8 nodes	1728	5.53	.69
	144	5.45	.68
	12	4.82	.60
4 nodes	1728	1.86	.47
	144	1.86	.47
	12	1.82	.46

successful in reducing the overall execution time to propagate several satellites. Table 4 shows the speedup and efficiency of  $P^3T$  for a various number of satellites. As seen in Table 4, the speedup achieved using all eight nodes of the hypercube was approximately three times larger than the speedup achieved using four nodes. With this parallel algorithm using six “working” nodes for an eight processor hypercube and only two “working” nodes for a four processor hypercube, an increase in speedup by approximately a factor of three was expected. In other words, since two processors are tied to input/output and cannot be used for computation, one should expect the gain to increase until we recover the loss of those two. More notable was the increase in efficiency using eight versus four nodes. The efficiency increased from .45 to .67. This increase in efficiency indicates that  $P^3T$  applied to a hypercube of greater dimension could yield even greater speedup and efficiency.

Table 4 also indicates that  $P^3T$  performance increased somewhat with an increase in the total number of satellites propagated. Because with this parallel algorithm the computation to communication ratio does not vary with the number of satellites, this small increase in performance must be primarily due to the diminishing impact of the algorithm’s overhead on total execution time. This overhead includes one additional message containing the total number of satellites to propagate from the distributing node to the other nodes; some small computations by working nodes to determine number of data sets to receive; and a halting message sent by the collecting node to the host once all of the nodes are finished. Because these additional messages and computations are only completed once in the program, the time cost associated with this overhead becomes negligible as the number of satellites propagated is increased. The speedup and efficiency remained fairly constant for greater than 144 satellites.

The performance results of this algorithm using only four and eight nodes indicated a potential increase in both speedup and efficiency if this algorithm could be applied to a hypercube of greater dimension. Because the number of working nodes is not fixed for this algorithm,  $P^3T$  could be applied easily to any size hypercube with no modifications.

The efficiency of the algorithm should increase with the number of processors until the time to distribute a separate satellite data to each working node exceeds the time required by a node to propagate a single satellite. A model was used to estimate the optimal number of nodes. The total execution time for  $P^3T$  to propagate  $n$  satellites with  $p$  processors,  $t(p)$ ,

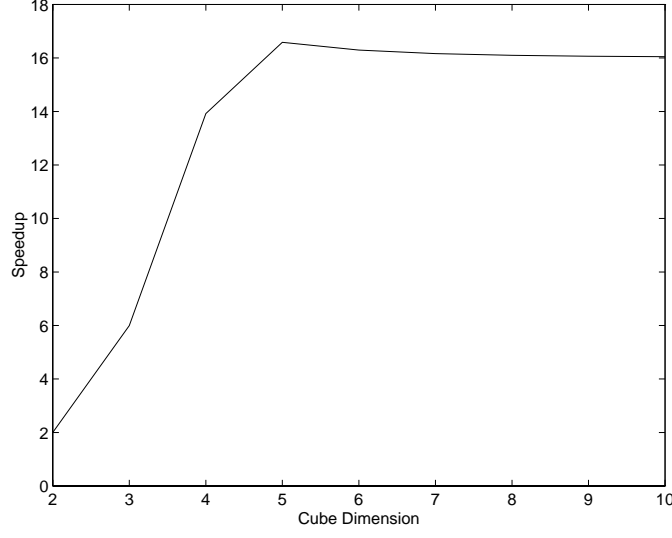


Figure 2: Theoretical Speedup for Propagating 2044 Satellites

can be modelled by

$$t(p) = t_{w1}(p) + t_{w2}(p) + t_c(p) ,$$

where  $t_{w1}(p)$  is the time the last node must wait to receive its first satellite data set,  $t_{w2}(p)$  is the total time the last node must wait to receive all of its subsequent satellite data sets, and  $t_c(p)$  is the computation time for each node to propagate its share of the  $n$  satellites. For this algorithm, there are  $p - 2$  working nodes. Denoting the time to send a single message between the distributing node and a working node as  $t_m(1)$ , the  $t_{w1}(p)$  may be modeled by the following:

$$t_{w1}(p) = (p - 3)t_m(1)$$

where  $t_m(1)$  denotes the time to send a single message between the distributing and working nodes. For the iPSC/2 it was found that

$$t_m(1) \cong .693 \text{ msec} .$$

The wait time  $t_{w2}$  is zero unless the number of working nodes is large enough, i.e.

$$t_{w2}(p) = \begin{cases} 0 & t_{w1}(p) \leq t_1 \\ (t_{w1}(p) - t_1) \left[ \frac{n}{p-2} - 1 \right] , & t_{w1}(p) > t_1 \end{cases}$$

where  $t_1$  is the computation time to propagate a single satellite (11.2 msec). Note that the factor  $\frac{n}{p-2} - 1$  is the number of subsequent satellite data sets. The computation time  $t_c$  is given by

$$t_c(p) = \frac{n}{p-2} t_1 .$$

Therefore, the speedup and efficiency are given by

$$S_p = \frac{n * t_1}{t_{w1}(p) + t_{w2}(p) + t_c(p)} ,$$

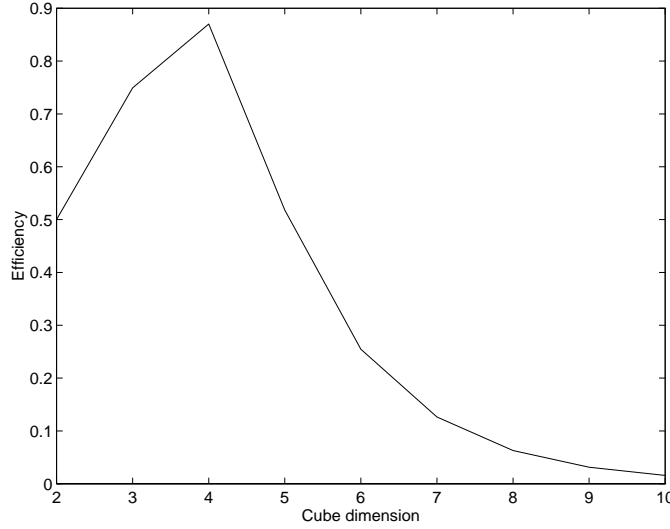


Figure 3: Theoretical Efficiency for Propagating 2044 Satellites

$$E_p = \frac{n * t_1}{p [t_{w1}(p) + t_{w2}(p) + t_c(p)]}.$$

Figures 2 and 3 depict these theoretical estimates of  $S_p$  and  $E_p$  for propagating 2044 satellites using 4 to 1024 processors. Using the above model,  $P^3T$  is capable of achieving a maximum speedup of 13.95 and an efficiency of 0.87 using 16 nodes.

## Conclusions

In this paper we have developed two ideas, control decomposition and domain decomposition, to parallelize the NAVSPASUR satellite motion model. The control decomposition idea is not efficient because the model is not computationally intensive enough. The domain decomposition can reach an efficiency of 87% when using a 16 - node hypercube. There are many orbit models in use nowadays. Several questions can be raised as a result of this research. How should an orbit theory be organized to take an advantage of MIMD computers? How should a semianalytic theory be organized for parallel computers? We are now working on a parallel version for the analytical model SGP4 in use by USSPACECOM and for the semianalytic satellite model developed at Draper Laboratory.

## Acknowledgements

This research was conducted for the office of Naval Research. The last two authors were supported in part by the Naval Postgraduate School. The authors would like to thank the referees for their valuable comments.

## References

1. Hwang, K. and Briggs, F. A., Computer Architecture and Parallel Processing, McGraw - Hill, New York, 1984.

2. Flynn, M. J., Very High - Speed Computing Systems, Proc. IEEE, Vol. 54, 1966, pp. 1901-1909.
3. Quinn, M. J., Designing Efficient Algorithms for Parallel Computers, McGraw - Hill, New York, 1987.
4. Phipps, W. E., Parallelization of the Naval Space Surveillance Center (NAVSPASUR) Satellite Motion Model, M.S. thesis, Naval Postgraduate School, Monterey, CA, 1992.
5. iPSC/2 User's Guide, INTEL Corp., order number 311532-006, 1990.

## **List of Figures**

1.  $P^3T$  algorithm
2.  $P^3T$  execution times
3. Theoretical speedup for propagating 2044 satellites
4. Theoretical efficiency for propagating 2044 satellites